REMARKS

Claims 1-22 remain pending in the application.

Claims 1-22 stand rejected under 35 U.S.C. 103(a) as being unpatentable over US Patent No. 6,438,536 issued to Edwards et al. (hereinafter Edwards) in view of US Patent No. 6,105,033 issued to Levine.

Examiner stated that, "with respect to claim 1, Edwards discloses determining from the access plan an executable function associated with a first operation code (see fig. 2B, the access plan is determined by the processing of optimizer component via execution functions associated with SQL statements as operation codes, which are generated by code generation component: col. 5, lines 32-64).

augmenting said first operation code in the access plan with a pointer to said executable function (the value of pointers in the operation code as a augment or parameter of a call function or subroutine, which is generated from SQL query code generation component or RAM CODEGEN executor: col. 4, lines 66-67 and col. 5, lines 1-10 and lines 40-56; also see col. 7, lines 60-67 and col. 8, lines 30-38)."

The Applicant respectfully objects to this misinterpretation because Edwards does not have the quoted language. It is the language taken verbatim from the claim 1 of the present invention, with Edwards reference column and line numbers added in brackets.

Examiner further stated that "Edwards discloses a query optimizer, Ram CODEGEN executor and code generation component for generating an access plan associated with the search queries entered by user and execution function such as call functions or call subroutines associated with SQL statements being as operation codes."

All independent claims 1, 10 and 19 of the present invention are specifically directed to show an improvement of a standard database management system which includes implementation of a direct call mechanism replacing the lookup function of a run-time interpreter and a method for pre-processing an already created access plan to provide a direct call mechanism in such a

system. They recite novel structure and thus distinguish over the cited prior art, under 35 U.S.C. 103(a). This is described in Figs. 1-4 and Specification on p. 2, li. 16-31; p. 3, li. 2-10, p. 6, li. 14-30, pages 7-9, and is used to provide faster access which is cost-effective.

Specifically, the independent claim 1 states:

1. A method for **pre-processing an access plan** generated for a query in a relational database management system to include a direct call mechanism replacing a lookup function of a run-time interpreter, said access plan including a plurality of operation codes, each of said operation codes being associated with one or more executable functions for performing the query, said method comprising the steps of:

(a) determining **from the access plan** an executable function associated with a first operation code; and

(b) augmenting said first operation code in the access plan with a **pointer** to said executable function to provide a direct call mechanism replacing a lookup function of a run-time interpreter.

As can be see from the bolded terms, claimed method is for **pre-processing** of an **existing access plan** which has op. codes and is to be interpreted by a Software Interpreter (not a Compiler). Step (a) determines, for an op. code from the existing access plan, an associated executable function which should be interpreted if there is no present invention. Step (b) augments the op. code, inside the existing access plan, and disallows interpretation of the executable function by the Interpreter because it replaces the op. code with **a direct pointer to the function**, the function being located outside the access plan, which implements the operation indicated by the op. code, thus removing the interpretive step. This means that later on, when the access plan is being executed at run-time, instead of interpreting the op. code, the pointer is used to call the replacement code, thus a term "direct call", which is described in much more detail in the Specification.

Thus, the claims are directed to preprocessing, which happens prior to execution, which will allow a run-time improvement of execution of an existing and previously optimized access plan whose steps have already been determined and are not changed by the present invention but are

substituted with pointers (Spec. p. 7, li. 5-10) in order to increase the run-time speed. As can be seen on p. 7, li. 25, the **pre-processing is performed by the Access Plan Manager prior to storing the improved access path plan into a memory cache and before the execution**. As can be seen on p.8, li. 14-16, this processed code section has pointers and is ready for execution, described on p. 8, li. 17-29.

It is true that Edwards in col. 4, li. 66-67, Fig. 2B, col. 5, li. 1-10, li 32-64, col. 7, li. 60-67 and col. 8, li. 30-38 mentions code generation component, CODEGEN executor, SQL query optimizer, pointers and call functions. These components are mentioned in most database applications. However, Edwards reference is from a different field and performs a different function. It is directed to a method used to **dynamically** generate **different, performance enhancing routines**, which will, during query **execution** time, decide which execution routines to use, upon certain run-time conditions. This is shown in Abstract, col. 2, li. 50-61. According to col. 2, li. 62 to col. 3, li. 18, during code generation, the calls to these different, performance enhancing record file management (RFM) layer routines are inserted into the generated code, instead of normally included routine calls, in order to eliminate record management function layer. This is confirmed in col. 5, li. 1-11, col. 6, li. 40-52 and col. 7, li. 32-61. The term "during execution" is repeated numerous times, i.e., in col. 5, li. 8, col. 7, li. 32. Col. 7, li. 50-54 show that the enhances new routine decides to bypass the RFM layer and in col. 7, li. 58 the reference mentions that it is the "run-time decision". In col. 9, li. 1-3 it is shown that the reference reduces the number of calls to the RFM layer and I/O layer. In col. 10, li. 1-8 it confirms this and declares that the reference relates to systems that generate executable code at execution time. Further, pointer, as defined in col. 7, li. 60-67 and col. 8 li. 30-38 is not pointing to a function, as in the present invention, but to a part of a table row where data are stored.

Moreover, Edwards reference does not have any and all elements from the independent claims of the present invention. It does **not include pointers to functions**, does not use the normal executable function from an already generated access plan (step (a)), does not create a new access plan with pointers to the same executable function (step (b)) and is not directed to a pre-processing method replacing a lookup function of a run-time interpreter.

It is noted with appreciation that Examiner held that Edwards does not explicitly teach a direct call mechanism replacing a lookup function of a run-time interpreter. The Examiner held that Levine discloses code generation, from which operation codes are generated is having call functions routines, which are interpreted as a pointer to a generated code segment (col. 8, lines 15-19).

Levine reference is from a different field of obsolete cache entries removal and has no elements of the claims of the present invention except code pointers. It is true that Levine reference uses pointers. A pointer is an often-used software tool which points to a code or data, depending on its function. However, in Levine reference a pointer points to the code to be deleted and not executed. Pointers in the present invention point to a function located outside the access plan which are to be executed. Col. 8, li. 15-19 of Levine is directed to the Cache Manager calls routines which provide a "code token" as an input which is "interpreted as a pointer" to a generated code segment or a data structure. The present invention uses a direct pointer which does not need interpretation. Thus, Levine teaches away from the present invention and it does not explicitly teach a direct call mechanism replacing a lookup function of a run-time interpreter.

Further, the Examiner stated that it would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teachings of Edwards with the teachings of Levine so as **to obtain a call function for being interpreted in the execution time** (Levine - col. 8, lines 15-19). This combination would have made the method for having code generation component **to generate call function routines** (Levine - Col. 8, lines 38-52) associated with execution functions and SQL statements (Levine - col. 3, lines 7-25) and during process of generating code for a specific SQL query, the code generation component produce calls to the difference performance enhancing into the generation code and this is easy to implement without having to make changes to the other components or layers of the relational database manager (Edwards - col. 45-67 and col. 3, lines 1-18).

The independent claims of present invention are not directed to "**obtaining a call function for being interpreted in the execution time**" but to use of pointers to **avoid** interpretation. They do not teach to "generate call function routines" because they already exist. The Levine reference,

according to col. 8, lines 38-[redacted]col. 3, li. 7-25 and throughout, teach[redacted]bsolete code deletion. Moreover, Edwards does not have col. 45-67 and col. 3, li. 1-18 explicitly teaches avoidance of the RFS layer, thus teaching away from the present invention. Moreover, these references are from completely different fields unrelated to a pre-processing method using pointers and the methods taught by these two references are among themselves from completely different art fields and cannot be combined. Moreover, it is shown that they do not perform any elements of the independent claims 1, 10, and 19 and therefore their dependent claims. Further, they satisfy a different need from a different area and do not teach pre-processing of existing access plan. Therefore, these references cannot be used to invalidate independent claims 1, 10, and 19 and their dependent claims. Because none of the referenced prior art teaches elements (a) to (b) of claims 1, 10 and 19, which are the main steps of the present invention, their combination is not a valid reason for rejection of these independent claims and claims dependent thereof. Therefore, each cited reference, by itself or in combination, cannot be used to invalidate claims 1, 10 and 19 because they fail to teach any and all the steps of these claims.

2. Examiner stated that claim 2 stands rejected because Edwards discloses the remaining operation codes in the access plan (col. 5, lines 1-5 and lines 40-56; ; also see col. 2, lines 3-15).

The referenced language in Edwards is directed to prior art and query execution with enhanced functions, which is not claimed in claim 2 of the present invention, which mentions that steps from claim 1 are repeated for all steps in the access plan.. Moreover, it is shown that this reference does not perform any elements of the independent claims 1, 10, and 19, and therefore their dependent claims. Further, it satisfies a different need from a different area and does not teach pre-processing of existing access plans to use pointers. Therefore, this reference cannot be used to invalidate the dependent claim 2.

3. With respect to claims 3-4, Examiner stated that Edwards discloses a method for pre-processing an access plan as discussed in claim 1 and that Edwards discloses a query optimizer, Ram CODEGEN executor and code generation component for generating an access plan associated with the search queries entered by user and execution function such as call functions or call subroutines associated with SQL statements being as operation codes.

However, as shown above, Edwards does not pre-process an existing access plan according to the independent claims of the present invention.

It is further held that Edwards does not explicitly teach a data structure for storing a pointer to said execution function and storing information associated with said executable function but that Levine discloses a data structure that contains the pointers to the code segment (col. 8, lines 15-19, and lines 65-67 and col. 9, lines 1-12) and executable functions (col. 8, lines 40-52).

As shown above, Levine reference is from a different field of obsolete cache entries removal and has no elements of the claims of the present invention except pointers. It is true that Levine reference uses code pointers. However, col. 8, li. 15-19 of Levine is directed to the Cache Manager calls routines which provide a "code token" as an input which is "interpreted as a pointer" to a generated code segment or a data structure. The present invention uses a direct pointer which does not need interpretation. Thus, Levine teaches away from the present invention and it does not explicitly teach a direct call mechanism replacing a lookup function of a run-time interpreter. Col. 8, li. 65-67, col. 9, lines 1-12 and col. 8, lines 40-52 are directed to cache with indirect pointers and deletion of obsolete functions.

Further, the Examiner stated that it would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teachings of Edwards with the teachings of Levine so as to obtain a data structure for pointers for allowing a process to indirectly reference locations within another data segment through pointers contained within the linkage segment (Levine - col. 9, lines 1-50, a call function for being **interpreted** in the execution time (Levine - col. 8, lines 15-19). This combination would have made the method for having code generation component **to generate call function routines** (Levine - col. 8, lines 38-52) associated with execution functions and SQL statements (Levine - col. 3, lines 7-25) and during process of generating code for a specific SQL query, the code generation component produce calls to the difference performance enhancing into the generation code and this is easy to implement without having to make changes to the other components or layers of the relational database manager (Edwards - col. 45-67 and col. 3, lines 1-18).

The independent claims of present invention are not directed to "**obtaining a call function for being interpreted in the execution time**" but to use of pointers to **avoid** interpretation. They do not teach to "generate call function routines" because they already exist. The Levine reference, according to col. 8, lines 38-52, col. 3, li. 7-25 and throughout, teaches obsolete code deletion. Moreover, Edwards does not have col. 45-67 and col. 3, li. 1-18 explicitly teaches avoidance of the RFS layer, thus teaching away from the present invention. Moreover, these references are from completely different fields unrelated to a pre-processing method using pointers and the methods taught by these two references are among themselves from completely different art fields and cannot be combined. Moreover, it is shown that they do not perform any elements of the independent claims 1, 10, and 19 and therefore their dependent claims. Further, they satisfy a different need from a different area and do not teach pre-processing of existing access plan. Therefore, these references cannot be used to invalidate independent claims 1, 10, and 19 and their dependent claims. Because none of the referenced prior art teaches elements (a) to (b) of claims 1, 10 and 19, which are the main steps of the present invention, their combination is not a valid reason for rejection of these independent claims and claims dependent thereof. Therefore, each cited reference, by itself or in combination, cannot be used to invalidate dependent claims 3 and 4 because they fail to teach any and all the steps of these claims.

4. With respect to claim 5, Examiner stated that Edwards discloses a method for pre-processing an access plan as discussed in claim 1 and that Edwards discloses augmenting said first operation code in the access plan with a second pointer (the value of pointers in the operation code as a augment or parameter of a call function or subroutine: Col. 4, lines 66-67 and Col. 5, lines 1-10 and lines 40-56; also see Col. 7, lines 60-67 and Col. 8, lines 30-38).

Edwards does not have the quoted language mentioning second pointer. The referenced columns are described above and cannot be used to invalidate claim 5 of the present invention.

The Examiner further held that Edwards discloses a query optimizer, Ram CODEGEN executor and code generation component for generating an access plan associated with the search queries entered by user and execution function such as call functions or call subroutines associated with SQL statements being as operation codes, and that Edwards does not explicitly teach a data

structure for storing a pointer to said execution function and storing information associated with said executable function but that, however, Levine discloses a data structure that contains the pointers to the code segment (Col. 8, lines 15-19, and lines 65-67 and Col. 9, lines 1-12) and executable functions (Col. 8, lines 40-52) and that, therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teachings of Edwards with the teachings of Levine so as to obtain a data structure for pointers for allowing a process to indirectly reference locations within another data segment through pointers contained within the linkage segment (Levine - Col. 9, lines 1-50, a call function for being **interpreted** in the execution time (Levine - Col. 8, lines 15-19) and that this combination would have made the method for having code generation component **to generate call function routines** (Levine - Col. 8, lines 38-52) associated with execution functions and SQL statements (Levine - Col. 3, lines 7-25) and during process of generating code for a specific SQL query, the code generation component produce calls to the difference performance enhancing into the generation code and this is easy to implement without having to make changes to the other components or layers of the relational database manager (Edwards - col. 45-67 and col. 3, lines 1-18).

The referenced columns are described above and cannot be used to invalidate claim 5 of the present invention. Moreover, these references are from completely different fields unrelated to a pre-processing method using pointers and the methods taught by these two references are among themselves from completely different art fields and cannot be combined. Moreover, it is shown that they do not perform any elements of the independent claims 1, 10, and 19 and therefore their dependent claims. Further, they satisfy a different need from a different area and do not teach pre-processing of existing access plan. Therefore, these references cannot be used to invalidate independent claims 1, 10, and 19 and their dependent claims. Because none of the referenced prior art teaches elements (a) to (b) of claims 1, 10 and 19, which are the main steps of the present invention, their combination is not a valid reason for rejection of these independent claims and claims dependent thereof. Therefore, each cited reference, by itself or in combination, cannot be used to invalidate dependent claim 5 because they fail to teach any and all the steps of this claim.

5. Examiner stated that claim 6 stands rejected because Edwards discloses assessing the executable function associated with the first operation code and if applicable, replacing the call to the executable function with a call to a second executable function (col. 5, lines 6-10, col. 6, lines 40-45 and col. 10, lines 1-8).

The referenced language in Edwards is directed to prior art and query execution with enhanced functions which replace the RFM layer functions, which is not claimed in claim 6 of the present invention. Moreover, it is shown that this reference does not perform any elements of the independent claims 1, 10, and 19, and therefore their dependent claims. Further, it satisfies a different need from a different area and does not teach pre-processing of existing access plans to use pointers. Therefore, this reference cannot be used to invalidate the dependent claim 6.

6. Examiner stated that claim 7 stands rejected because Edwards discloses intermediate function which includes processing operations for the first operation code or the executable function associated with the first operation code (col. 5, lines 5-32 and lines 40-55).

The referenced language in Edwards is directed to prior art and query execution with enhanced functions which replace the RFM layer functions, which is not claimed in claim 7 of the present invention. Moreover, it is shown that this reference does not perform any elements of the independent claims 1, 10, and 19, and therefore their dependent claims. Further, it satisfies a different need from a different area and does not teach pre-processing of existing access plans to use pointers. Therefore, this reference cannot be used to invalidate the dependent claim 7.

7. Examiner stated that claims 8-9 stand rejected because Edwards discloses a method for preprocessing an access plan as discussed in claim 1 and that Edwards discloses a query optimizer, Ram CODEGEN executor and code generation component for generating an access plan associated with the search queries entered by user and execution function such as call functions or call subroutines associated with SQL statements being as operation codes but that Edwards does not explicitly teach gathering statistics on the use of the executable function; and a pause for receiving user input before or after the call to the executable function and that, however, Levine discloses a **analyzing each SQL statement** entered by user for defining the

access plan, which will produce the optimum performance for the execution of the statement (user enter queries into the database in order to obtain or extract requested data, and the query optimizer analyzes how best to conduct the users' query of the database in terms of optimum speed in accessing the requested data: col. 5, lines 35-45 and lines col. 1, lines 25-30 and lines 38-45; also see col. 18, lines 28-32) and that, therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teachings of Edwards with the teachings of Levine so as to obtain a way to analyze the **SQL statement, which are entered by user** (col. 5, lines 35-45 and col. 6, lines 15-25). This combination would have made the method for having routines for **analyzing each query statement** (Levine col. 6, lines 15-26), a code generation component to generate call function routines (Levine - col. 8, lines 38-52) associated with execution functions and SQL statements (Levine - col. 3, lines 7-25) and during process of generating code for a specific SQL query, the code generation component produce calls to the difference performance enhancing into the generation code and this is easy to implement without having to make changes to the other components or layers of the relational database manager (Edwards - col. 45-67 and col. 3, lines 1-18).

Claim 8 of the present invention is directed to gathering **statistics** on use of the executable function. Claim 9 of the present invention is directed to **inserting a pause for receiving user's input**.

It is noted with gratitude that the Examiner held that Edwards does not explicitly teach gathering statistics on the use of the executable function; and a pause for receiving user input before or after the call to the executable function. However, the fact analyzes each whole **SQL statement** does not mean that it teaches gathering statistics on the **use of each one executable function**. Moreover, the Office action is mute regarding inserting a deliberate pause for receiving **user input** before or after the call to the **executable function** but talks about **inputting the whole SQL statement.**

The referenced columns are described above and cannot be used to invalidate claims 8-9 of the present invention. Moreover, these references are from completely different fields unrelated to a pre-processing method using pointers and the methods taught by these two references are among

themselves from completely different art fields and cannot be combined. Moreover, it is shown that they do not perform any elements of the independent claims 1, 10, and 19 and therefore their dependent claims. Further, they satisfy a different need from a different area and do not teach pre-processing of existing access plan. Therefore, these references cannot be used to invalidate independent claims 1, 10, and 19 and their dependent claims. Because none of the referenced prior art teaches elements (a) to (b) of claims 1, 10 and 19, which are the main steps of the present invention, their combination is not a valid reason for rejection of these independent claims and claims dependent thereof. Therefore, each cited reference, by itself or in combination, cannot be used to invalidate dependent claims 8-9 because they fail to teach any and all the steps of these claims.

8. Examiner stated that Claim 10 is essentially the same as claim 1 except that it is directed to a computer program product rather than a method ('536 of see fig. 2B, the access plan is determined by the processing of optimizer component via execution functions associated with SQL statements as operation codes, which are generated by code generation component: col. 5, lines 32-64; and the value of pointers in the operation code as a augment or parameter of a call function or subroutine, which is generated from SQL query code generation component or RAM CODEGEN executor: col. 4, lines 66-67 and col. 5, lines 1-10 and lines 40-56; also see col. 7, lines 60-67 and col. 8, lines 30-38, '033 of col. 8, lines 15-19), and is rejected for the same reason as applied to the claim 1 hereinabove.

Applicant respectfully points to his argument from above, regarding claim 1, 10 and 19, which shows that each cited reference, by itself or in combination, cannot be used to invalidate independent claims 1, 10 and 19 because they fail to teach any and all the steps of these claims.

9. Examiner stated that Claim 11 is essentially the same as claim 2 except that it is directed to a computer program product rather than a method (col. 5, lines 1-5 and lines 40-56; ; also see col. 2, lines 3-15), and is rejected for the same reason as applied to the claim 2 hereinabove.

Applicant respectfully points to his argument from above, especially regarding claim 2 and claims 1, 10 and 19, which shows that each cited reference, by itself or in combination, cannot be

used to invalidate independent claims 1, 10 and 19 and dependent claim 11 because they fail to teach any and all the steps of these claims.

10. Examiner stated that Claim 12 is essentially the same as claim 3 except that it is directed to a computer program product rather than a method (col. 8, lines 15-19, and lines 65-67 and col. 9, lines 1-12), and is rejected for the same reason as applied to the claim 3 hereinabove.

Applicant respectfully points to his argument from above, especially regarding claim 3 and claims 1, 10 and 19, which shows that each cited reference, by itself or in combination, cannot be used to invalidate independent claims 1, 10 and 19 and dependent claim 12 because they fail to teach any and all the steps of these claims.

11. Examiner stated that Claim 13 is essentially the same as claim 4 except that it is directed to a computer program product rather than a method (col. 8, lines 40-52), and is rejected for the same reason as applied to the claim 4 hereinabove.

Applicant respectfully points to his argument from above, especially regarding claim 4 and claims 1, 10 and 19, which shows that each cited reference, by itself or in combination, cannot be used to invalidate independent claims 1, 10 and 19 and dependent claim 13 because they fail to teach any and all the steps of these claims.

12. Examiner stated that Claim 14 is essentially the same as claim 5 except that it is directed to a computer program product rather than a method ('536 of the value of pointers in the operation code as a augment or parameter of a call function or subroutine: col. 4, lines 66-67 and col. 5, lines 1-10 and lines 40-56; also see col. 7, lines 60-67 and col. 8, lines 30-38; and '033 of col. 8, lines 15-19, and lines 65-67 and col. 9, lines 1-12, and col. 8, lines 40-52), and is rejected for the same reason as applied to the claim 5 hereinabove.

Applicant respectfully points to his argument from above, especially regarding claim 5 and claims 1, 10 and 19, which shows that each cited reference, by itself or in combination, cannot be used to invalidate independent claims 1, 10 and 19 and dependent claim 14 because they fail to teach any and all the steps of these claims.

13. Examiner stated that Claim 15 is essentially the same as claim 6 except that it is directed to a computer program product rather than a method (col. 5, lines 6-10, col. 6, lines 40-45 and col. 10, lines 1-8), and is rejected for the same reason as applied to the claim 6 hereinabove.

Applicant respectfully points to his argument from above, especially regarding claim 6 and claims 1, 10 and 19, which shows that each cited reference, by itself or in combination, cannot be used to invalidate independent claims 1, 10 and 19 and dependent claim 15 because they fail to teach any and all the steps of these claims.

14. Examiner stated that Claim 16 is essentially the same as claim 7 except that it is directed to a computer program product rather than a method (col. 5, lines 5-32 and lines 40-55), and is rejected for the same reason as applied to the claim 6 hereinabove.

Applicant respectfully points to his argument from above, especially regarding claim 7 and claims 1, 10 and 19, which shows that each cited reference, by itself or in combination, cannot be used to invalidate independent claims 1, 10 and 19 and dependent claim 16 because they fail to teach any and all the steps of these claims.

15. Examiner stated that Claims 17-18 are essentially the same as claims 8-9 except that they are directed to a computer program product rather than a method (user enter queries into the database in order to obtain or extract requested data, and the query optimizer analyzes how best to conduct the users' query of the database in terms of optimum speed in accessing the requested data: col. 5, lines 35-45 and lines col. 1, lines 25-30 and lines 38-45; also see col. 18, lines 28-32), and are rejected for the same reason as applied to the claims8-9 hereinabove.

Applicant respectfully points to his argument from above, especially regarding claims 8-9 and claims 1, 10 and 19, which shows that each cited reference, by itself or in combination, cannot be used to invalidate independent claims 1, 10 and 19 and dependent claims 17-18 because they fail to teach any and all the steps of these claims.

16. Examiner stated that Claim 19 is essentially the same as claim 1 except that it is directed to a system rather than a method ('536 of see fig. 2B, the access plan is determined by the processing of optimizer component via execution functions associated with SQL statements as operation codes, which are generated by code generation component: col. 5, lines 32-64; and the value of pointers in the operation code as a augment or parameter of a call function or subroutine, which is generated from SQL query code generation component or RAM CODEGEN executor: col. 4, lines 66-67 and col. 5, lines 1-10 and lines 40-56; also see col. 7, lines 60-67 and col. 8, lines 30-38; '033 of col. 8, lines 15-19), and is rejected for the same reason as applied to the claim 1 hereinabove.

Applicant respectfully points to his argument from above, regarding claims 1, 10 and 19, which shows that each cited reference, by itself or in combination, cannot be used to invalidate independent claims 1, 10 and 19 because they fail to teach any and all the steps of these claims.

17. Examiner stated that, with respect to claims 20-21, Edwards discloses a relational database system as discussed in claim 19, that Edwards discloses a query optimizer, Ram CODEGEN executor and code generation component for generating an access plan associated with the search queries entered by user and execution function such as call functions or call subroutines associated with SQL statements being as operation codes. Edwards does not explicitly teach a data structure for storing a pointer to said execution function and storing information associated with said executable function, but, however, Levine discloses a data structure that contains the pointers to the code segment (col. 8, lines 15-19, and lines 65-67 and col. 9, lines 1-12) and executable functions (col. 8, lines 40-52) and that, therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teachings of Edwards with the teachings of Levine so as to obtain a data structure for pointers for allowing a process to indirectly reference locations within another data segment through pointers contained

within the linkage segment (Levine - col. 9, lines 1-50, a call function for being **interpreted** in the execution time (Levine - col. 8, lines 15-19). This combination would have made the method for having code generation component **to generate call function routines** (Levine - col. 8, lines 38-52) associated with execution functions and SQL statements (Levine - col. 3, lines 7-25) and during process of generating code for a specific SQL query, the code generation component produce calls to the difference performance enhancing into the generation code and this is easy to implement without having to make changes to the other components or layers of the relational database manager (Edwards - col. 45-67 and col. 3, lines 1-18).

Applicant respectfully points to his argument from above, especially regarding dependent claim 5 and claims 1, 10 and 19, which shows that each cited reference, by itself or in combination, cannot be used to invalidate independent claims 1, 10 and 19 because they fail to teach any and all the steps of these claims and the dependent claims 20-21.

18. Examiner stated that, with respect to claim 22, Edwards discloses a method for pre-processing an access plan as discussed in claim 1, that Edwards discloses a query optimizer, Ram CODEGEN executor and code generation component for generating an access plan associated with the search queries entered by user and execution function such as call functions or call subroutines associated with SQL statements being as operation codes, that Edwards does not explicitly teach a data structure for storing a pointer to said execution function and storing information associated with said executable function but that, however, Levine discloses a data structure that contains the pointers to the code segment (col. 8, lines 15-19, and lines 65-67 and col. 9, lines 1-12) and executable functions (col. 8, lines 40-52) and that, therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teachings of Edwards with the teachings of Levine so as to obtain a data structure for pointers for allowing a process to indirectly reference locations within another data segment through pointers contained within the linkage segment (Levine - col. 9, lines 1-50, a call function for being **interpreted** in the execution time (Levine - col. 8, lines 15-19) and that this combination would have made the method for having code generation component **to generate call function routines** (Levine - col. 8, lines 38-52) associated with execution functions and SQL statements (Levine - col. 3, lines 7-25) and during process of generating code for a specific

SQL query, the code generation component produce calls to the difference performance enhancing into the generation code and this is easy to implement without having to make changes to the other components or layers of the relational database manager (Edwards - col. 45-67 and col. 3, lines 1-18).

Applicant respectfully points to his argument from above, especially regarding dependent claim 5 and claims 1, 10 and 19, which shows that each cited reference, by itself or in combination, cannot be used to invalidate independent claims 1, 10 and 19 because they fail to teach any and all the steps of these claims and the dependent claim 22.

Regarding claims 1-22, none of the cited references teaches, shows, suggests or is even remotely related to pre-processing of existing access plans to augment op. code with pointers, as claimed by the present invention. Therefore, these reference cannot be used to invalidate independent claims 1, 10, and 19 and their dependent claims. Moreover, the Examiner combined references from different arts in order to reject claims 1-22, by quoting parts of sentences nonexistent in those references. However, even if these quotes are correct, the combination must be pointed to in the prior art itself and no such combination is pointed to in the cited references nor it could be since they are from different fields. Therefore, these references cannot be used to invalidate independent claims 1, 10 and 19 and their dependent claims because they fail to teach any and all the steps of these claims.

Therefore, all submitted claims are allowable over the cited references and their reconsideration is respectfully requested. Improper combination of cited references is used in each claim rejection in the Office Action. None of the cited references suggests combination under In re Sernaker, 217 U.S.P.Q. 1, 6 (CAFC 1983), and one skilled in the art would have no reason to make a combination since they are from different fields, impossible to combine and individually complete. Moreover, none of the cited references discloses the subject matter and features of claims 1-22 of the present invention and even if they did show some individual features, they would not be able to meet the claims of the present invention which provide new and unexpected results over these references and are thus unobvious and patentable under Sec. 103.

The prior art made of record and not relied upon but considered pertinent to Applicants' disclosure has been reviewed but found to be even less relevant than the Edwards and Levine references.

In view of the above, it is submitted that this application is now in good order for allowance, which applicant respectfully solicits. Should matters remain which the Examiner believes could be resolved in a telephone interview, the Examiner is kindly requested to telephone the Applicant's undersigned attorney. No additional fee is required in connection with this communication since the Response is mailed within three months from the Office Action and the number of claims is not extending the original number of claims. However, any underpayment is authorized to be charged to Deposit Account Number **09-0460** in the name of IBM Corporation.

Respectfully submitted,

Date: February 10, 2004

Sandra M. Parker
Reg. No. 36,233

LAW OFFICE OF SANDRA M. PARKER
329 La Jolla Avenue
Long Beach, CA 90803
Phone: (562) 597-7504
FAX: (562) 597-1841